# 程序设计方法学

CHENGXU SHEJI FANGFAXUE

（第2版）

胡正国 吴 健 邓正宏 编著

国防工业出版社
National Defense Industry Press

# 程序设计方法学

(第2版)

胡正国　吴 健　邓正宏　编著

国防工业出版社

·北京·

# 内 容 简 介

本书主要介绍程序设计方法学这一新兴学科的主要内容，即结构化程序、程序正确性证明、结构化程序的正确性证明、递归程序及其正确性证明、程序的形式推导技术、程序变换技术、面向对象的设计方法和大型程序设计方法学基础等。

本书可供大专院校计算机专业学生使用，也可供硕士研究生及从事计算机工作的科研人员参考。

**图书在版编目(CIP)数据**

程序设计方法学/胡正国，吴健，邓正宏编著. —2 版.
北京：国防工业出版社，2009.1
ISBN 978-7-118-06051-5

Ⅰ. 程… Ⅱ. ①胡…②吴…③邓… Ⅲ. 程序设计－方法 Ⅳ. TP311.11

中国版本图书馆 CIP 数据核字(2008)第 181011 号

※

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号 邮政编码 100048)
新艺印刷厂印刷
新华书店经售

*

**开本** 787×1092 1/16 **印张** 17 **字数** 394 千字
2009 年 1 月第 2 版第 1 次印刷 **印数** 1—4000 册 **定价** 26.00 元

**(本书如有印装错误，我社负责调换)**

国防书店：(010)68428422 发行邮购：(010)68414474
发行传真：(010)68411535 发行业务：(010)68472764

# 前　言

程序设计方法学是20世纪60年代末到70年代初形成和发展起来的计算机科学领域中的一个新兴学科。近年来，这一学科的发展比较迅速，取得了不少令人鼓舞的成果。为了介绍这一学科的一些基本内容，我们在多年进行这方面教学的基础上编写了这本教材。

由于编写这本教材的目的是向读者介绍这一学科的一些最基本的内容，因而在讲述时尽量避免一些严格的形式化系统。读者在掌握了本教材基本内容以后，可以较顺利地阅读其他有关的专著，以求对这一学科有更深入的了解。

另外，本教材中没有采用统一的语言描述程序。我们感到，这样做尽管从表面上看不够统一，但是可以使读者接触较多的控制结构，而且有利于读者阅读有关专著。因而，权衡利弊我们做出了这样的选择。

这本教材初次出版后，受到了许多兄弟院校的欢迎，并且提出了一些宝贵的意见和建议。在此我们向这些同志表示深深的谢意。这次除了对原有的内容做了一些修改和调整外，补充了面向对象的程序设计方法及大型程序设计方法学基础两章。本教材讲授时数约60学时，可供计算机专业高年级学生及硕士研究生使用，也可供计算机工作者参考。

本书由西北工业大学计算机科学与工程系的教师组织编著。其中，第1章到第3章、第5章到第6章及第8章由胡正国编著，第4章由邓正宏编著，第10章由吴健编著。第7章和第9章由厦门大学计算机科学系蔡经球编著。西安电子科技大学陈家正教授在百忙中认真审阅了全书的内容，并提出了宝贵的意见，在此表示衷心的感谢。

由于时间仓促，加之编者水平有限，不妥之处在所难免，诚恳希望同行的专家及广大读者提出宝贵意见。

编著者

2008年5月8日

# 目　录

# 第 1 章　程序设计方法学简介

## 1.1　程序设计方法学的产生

随着计算机硬件技术的不断发展，程序设计方法也在不断地改进和发展。在计算机发展的早期阶段，由于计算机硬件功能较弱，即运算速度较慢、存储空间较小，因而主要采用机器语言或汇编语言编写程序。这时，程序设计方法的重点是如何尽可能多地使用一些技巧，以节省内存空间，提高运算速度。在 20 世纪 50 年代末到 60 年代初虽然也相继出现了一些高级语言，例如 FORTRAN 语言、ALGOL 语言等，为提高程序员的算法表达能力，减轻一些繁琐的劳动创造了一定的条件。但是，由于在这一时期计算机主要应用于科学计算，而且程序的规模一般都比较小，因而从程序设计方法上看并没有发生根本的变化。总的来说，在这一阶段，程序设计被看做是一项技巧性很强的工作，也可以说，采用的是一种手工式的设计方法。

20 世纪 60 年代以来，随着硬件技术的迅猛发展和计算机应用领域的急剧扩大，不仅绝大多数计算机程序都采用高级语言编写，而且计算机的一些规模较大的应用软件也采用某些高级语言来编写。这时，由于一般要编写的程序的规模都比较大，因而对这些程序来说，运行时间和占存储空间的大小已经不是编写者要考虑的主要问题，而主要问题已经逐渐转化为希望编写出的程序结构清晰、容易阅读、容易修改、容易验证，即希望得到好结构的程序。要做到这一点，就必须改变传统的程序设计方法的观念，采用一种新的策略和方法来进行程序设计。这就是所谓的“软件工程”的方法，或者说工程化的设计方法。

程序设计的这一发展过程，可以简单地概括为：手工艺式的设计方法→工程化的设计方法。

程序设计方法的这一发展是方法论上的一个飞跃。为什么会产生这一飞跃呢？这是由于通常所说的“软件危机”引起的。所谓软件危机是指自 20 世纪 60 年代末到 70 年代初，随着计算机硬件技术的发展和计算机应用范围的不断扩大，要研制一些大的软件系统，例如，操作系统、程序库等。而一个大的系统的编制周期是较长的，工作量是很大的（常常需要用几百人·年到几千人·年），并且由于传统的程序设计方法的局限性，设计出的软件系统往往隐藏着许多错误，这就给软件的使用和维护带来很大的困难。一方面客观上需要研制大量的软件，另一方面按照原有的方法研制软件周期长、可靠性差、维护困难，这就是“危机”之所在。为了克服这一“危机”，一方面需要对程序设计方法、程序的可靠性等问题进行系统的研究，另一方面也需要对软件的编制、管理和维护的方法进行研究。这就是程序设计方法学产生的历史背景。

1968 年，E.W.Dijkstra 首先提出“GOTO 语句是有害的”，向传统的程序设计方法提出了挑战，从而引起了人们对程序设计方法讨论的普遍重视。许多著名的计算机科学家都参加了这种讨论。程序设计方法学这一学科也正是在这种广泛而深入的讨论中逐渐产生和形成的。

什么是程序设计方法学呢？简单地说，程序设计方法学是讲述程序的性质以及程序设计的理论和方法的一门学科。

由于程序设计方法学是一门新兴的发展较为迅速的学科，因而它的内容是比较丰富的。例如，结构程序设计、数据抽象与模块化程序设计、程序正确性证明、程序变换、程序的形式说明与推导、程序综合技术、面向对象的程序设计方法、大型程序设计方法学基础等。由于篇幅限制，本书仅介绍其中的一些主要内容。

在程序设计方法学中，结构程序设计占着十分重要的位置，可以说，程序设计方法学是在结构程序设计的基础上逐步发展和完善起来的。因而，了解和掌握结构程序设计的概念及它所讨论的一些主要问题对了解什么是程序设计方法学是很有帮助的。下面，首先简要介绍这方面的内容。

## 1.2 结构程序设计及其讨论的一些主要问题

什么是结构程序设计呢？到目前为止，虽然人们从不同的角度给出了不少的定义，但是还没有一个很严格的，又能为大家普遍接受的定义。

1974 年，D.Gries 教授将已有的对结构程序设计的不同解释归纳为 13 种。其中比较有代表性的有以下几种：

(1) 结构程序设计是避免用 GOTO 语句的一种程序设计；

(2) 结构程序设计是自顶向下的程序设计；

(3) 结构程序设计是一种组织和编制程序的方法，利用它编制的程序是容易理解和容易修改的；

(4) 程序结构化的一个主要功能是使得正确性的证明容易实现；

(5) 结构程序设计允许在设计过程中的每一步验证其正确性，即自动导致自我说明与自我捍卫的程序风格；

(6) 结构程序设计讨论了如何将任何大规模的和复杂的流程图转换成一种标准的形式，使得它们能够用几种标准的控制结构（通常是顺序、分支和重复）通过重复和嵌套来表示。

这些定义（或解释）从不同的角度反映了结构程序设计所讲述的主要问题。综合这些定义，可以使我们对结构程序设计有一个概貌性的了解。为了以后讲述方便起见，我们给出下面的定义：

结构程序设计是一种进行程序设计的原则和方法，按照这种原则和方法设计出的程序的特点是结构清晰、容易阅读、容易修改、容易验证。

按照结构程序设计的要求设计出的程序设计语言称为结构程序设计语言。利用结构程序设计语言，或者说按照结构程序设计的思想编制出的程序称为结构化程序，或者好

结构的程序。

下面，简要介绍结构程序设计所讨论的一些主要问题，在第 2 章还将对有些问题做进一步的介绍。

### 一、关于 GOTO 语句的问题

前面已经提到，关于程序设计方法的讨论是由避免使用 GOTO 语句引起的。因而，首先对这一问题做一些简单的介绍。

在 20 世纪 60 年代末到 70 年代初，关于 GOTO 语句的争论是比较激烈的。

主张从高级语言中去掉 GOTO 语句的人认为，GOTO 语句是对程序结构影响最大的一种有害的语句。他们的主要理由是 GOTO 语句使程序的静态结构与它的动态执行之间有很大的差别，这样使程序难以阅读、难以查错。对一个程序来说，人们最关心的是它运行的正确与否，去掉 GOTO 语句以后，可直接从程序结构上反映出程序运行的过程。这样，不仅使程序的结构清晰、便于阅读、便于查错，而且也有利于程序的正确性证明。

持不同意见的人们认为，GOTO 语句使用起来比较灵活，而且有些情形能提高程序的效率。如果一味强调删去 GOTO 语句，有些情形反而会使程序过于复杂，增加一些不必要的计算量。

1974 年，D．E.knuth 对于 GOTO 语句的争论做了全面的公正的评述。他的基本观点是：不加限制地使用 GOTO 语句，特别是使用往回跳的 GOTO 语句，会使程序结构难于理解，这种情形应尽量避免使用 GOTO 语句。另外，为了提高程序的效率，同时又不破坏程序的良好结构，有控制地使用一些 GOTO 语句是必要的。用他的话来说："有些情形，我主张废除转向语句；另外一些情形，则主张引进转向语句。"

进一步讲，GOTO 语句能不能消除呢？或者说 GOTO 语句这一语言成分能不能从程序设计语言中取消呢？回答是肯定的。1966 年，G.Jacopini 和 C.Bohm 从理论上证明了任何程序都可以用序列结构、条件结构和循环结构表示出来，即任何程序都可以用图 1.1 所示的 3 种结构表示出来（这一结论将在第 2 章证明）。

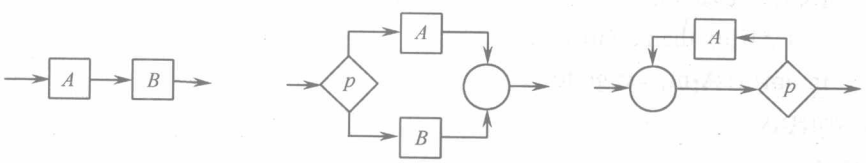


图 1.1

更进一步，如何从一个具体的程序中消除 GOTO 语句呢？通常可以采用增加辅助变量，或者改变程序执行顺序的方法来消除 GOTO 语句。

**例 1.1**　考查下面一段程序

```
L1: if B1 then go to L2 fi;
    S1;
    if B2 then go to L2 fi;
    S2;
    go to L1;
```

L2: $S_3$

需要指出的是，程序中符号 fi 和相应 if 配对使用，表示条件语句的结束。这个记号是由 D.E.Knuth 提出来的，采用它可以省略一些语法括号 begin…end。同时，使程序结构更为清晰。例如，语句

if p then begin $S_1$; $S_2$ end

可以表示为

if p then $S_1$; $S_2$ fi

语句

```
if p then begin S1; S2   end
        else begin S3; S4 end
```

可以表示为

```
if p then S1;S2 ; else S3; S4   fi
```

显然，例 1.1 中的程序段包含有 GOTO 语句。如果引入一个逻辑变量 $p$，则可以消除 GOTO 语句，建立下面的一段与它等价的程序

```
p:=true;
while p do
 if   B1   then p:=false;
    else S1; if B2 then p:=false;
    else S2 fi fi ;
S3;
```

**例 1.2**（查表程序）在一个表中有 $m$ 个不同的数 $A[1]$，$A[2]$，…，$A[m]$。现在，要编写一段程序，在该表中查找数 $x$。若找到 $x$，将该数以及它在表中的位置打印出来；否则，将该数加到这个表中去。这一工作可以由下面的一段程序来完成。

```
……
      for i:=1 to m do
        if A[i]=x then go to 1 fi;
      m:=m+1; A[m]:=x; go to 2;
1: write(i,x);
2:……
```

这段程序包含有两个 GOTO 语句，和例 1.1 类似，可以用增加辅助逻辑变量 $p$ 的方法消除 GOTO 语句，得到下面等价程序

```
……
p:=false;
for i:=1 to m do
 if A[i]=x then p:=true; y:=i fi;
if p then write(y,x);
     else m:=m+1; A[m]:=x fi;
……
```

另外，也可以用改变程序执行程序的方法消除 GOTO 语句，得到下面的等价程序

```
……
i:=1
while(A[i]≠x)∧(i<m) do i:=i+1;
if i >m then m:=m+1; A[m]:=x;
        else write(i,x) fi;
……
```

上面通过两个例子介绍了消除 GOTO 语句的一些方法。但是，对于比较复杂的程序，要用类似的方法往往是比较困难的。这时，常采用另一个比较行之有效的方法，即给语言中增加一些新的控制结构。例如，结构 FORTRAN 语言、结构 COBOL 语言等就是在原有的高级语言的基础上增加了若干个新的控制结构而建立的。另外，一些常用的结构程序设计语言（例如，PASCAL 语言、BLISS 语言），除了数据结构上作了若干新的扩充以外，也都分别设计了多种控制结构，为避免使用 GOTO 语句创造了条件。关于控制结构的研究，20 世纪 70 年代以来已经取得了不少成果，有关这方面比较详细的内容将在第 2 章中介绍。在这里需要指出的是，虽然关于结构程序设计的讨论是从废除 GOTO 语句开始的，但是，绝不能认为结构程序设计就是避免 GOTO 语句的程序设计方法。事实上，结构程序设计讨论的是一种新的程序设计的方法和风格，它所关注的焦点是所得到的程序结构好坏，而有无 GOTO 语句并不是一个程序结构好坏的标志。这也就是说，限制和避免使用 GOTO 语句是得到结构化程序的一个手段，而不是目的。

**二、程序的结构**

前面已经谈到，结构程序设计的目标是得到一个好结构的程序。再具体一些，好结构的程序是由一些什么样的语法结构组成的程序呢？一般地说，结构化程序是由下面七种结构组成的。

1．序列结构

序列结构如图 1.2 所示，图中 *A*、*B* 可以是一个语句（甚至是空语句），也可以是这里将要介绍的 7 种结构中的一种结构。当 *A* 或 *B* 是一个结构时，称它为这一序列结构的子结构。对下面几种结构亦有完全类似的情况。


图 1.2

2．选择结构

一般的两种选择结构如图 1.3 所示（它们通常也称为条件结构），特殊的选择结构如图 1.4 所示。

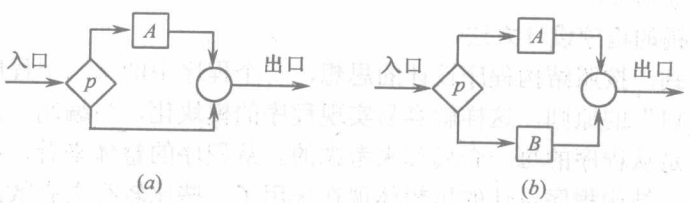


图 1.3

显然，图 1.3（$a$）的结构是图 1.3（$b$）的特殊情况，而图 1.4 所示的结构可以通过嵌套使用图 1.3 的结构表示出来，但是为了更灵活地编写程序，仍然将它们看做不同的结构。

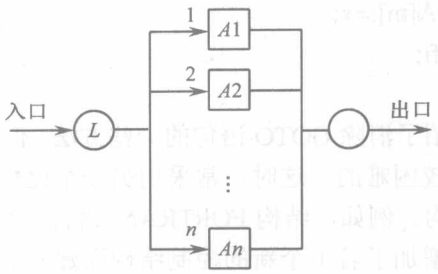


图 1.4

在以上 4 种结构（序列结构和 3 种选择结构）中，都没有出现将控制转移到本结构入口的情形，这样的结构通常称为开型结构。

3. 循环结构

循环结构有 3 种，如图 1.5 所示，这 3 种循环结构依次为 WHILE 循环、REPEAT 循环和 $N$+1/2 循环。和选择结构类似，虽然前两种循环可以看做是第 3 种循环的特殊情形，但为了构造程序方便，仍然将它们看做不同的结构。

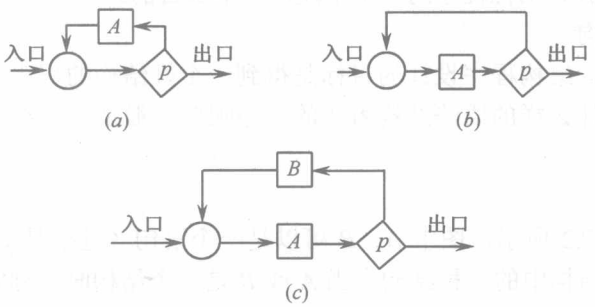


图 1.5

这 3 种循环结构虽然形式不同，但是它们有一个共同点，即在一定条件下，将控制转移到本结构的入口处，这样的结构通常称为闭型结构。

这 7 种结构都有一个很显著的特点，即每一种结构都严格遵循“一个入口，一个出口”的原则，这一原则是结构程序设计中的一个十分重要的原则。也正是由于遵循了这一原则，一个复杂的程序才可能分解为若干个结构以及若干层子结构，从而使程序的结构层次分明、清晰易懂。

**三、逐步求精的程序设计方法**

上面已经谈到，按照结构程序设计的思想，一个程序中的每一个程序段均服从“一个入口、一个出口”的原则，这样就容易实现程序的模块化，为编制、调试、验证程序提供了方便。这是从程序的每一个局部来考虑的。从程序的整体来看，或者说从程序编制的全过程来看，结构程序设计的思想体现在采用了一些比较行之有效的方法，在这些方法中比较有代表性的是“逐步求精”的方法。

所谓“逐步求精”的方法，就是在编制一个程序时，首先考虑程序的整体结构而忽视一些细节问题，然后逐步地、一层一层地细化程序直至用所选用的语言完全描述每一个细节，即得到所期望的程序为止。在编制过程中，一些算法可以采用编制者们所能共同接受的语言来描述（甚至用自然语言来描述）。

下面，通过一个有代表性的例子来说明这一方法的基本思想。

**例 1.3**　编写一个程序，打印出前 $N$ 个素数（$N$ 是给定的正整数）。

用 PASCAL 语言来编写这一程序。采用逐步求精的方法，编写过程可按以下步骤实现。

第一步：根据所提出的问题，程序的结构可以写为

```
program ex1(output);
  var i, x: integer;
  begin
    x:=1;
    for i:=1 to N do
    begin
      x:=“下一个素数”;
   write(x)
  end
end.
```

第二步：对语句 x：= “下一个素数”进一步求精，为此引进一个布尔变量 prim,这个语句可以用循环语句表示为

```
repeat x:=x+1;
  prim=“x 是一个素数”
until prim;
```

第三步：对语句 prim:= “x 是一个素数”进一步求精。“$x$ 是一个素数”即 $x$ 只能被 1 及它自身除尽，或者说 $x$ 不能被 2，3，…，$x$-1 整除。这样，这条语句可以用下面一段程序表示为

```
k:=2; prim:=true;
while prim AND(k≤lim) do
begin
  prim:= “x 不能被 k 整除”;
  k:=k+1;
end;
```

这里，lim 是 $k$ 的上界，即 lim=$x$-1。显然，当 $x$ 是一个素数时，这段程序执行结果布尔变量 prim 为真；当 $x$ 不是素数时，$x$ 总能被某一个 $k$ 整除，那么，这段程序执行结果布尔变量 prim 为假。

第四步：对语句 prim:=“x 不能被 k 整除”进一步求精。利用求余运算，这一语句可以表示为

```
prim:=(x MOD k)≠0
```

将这一语句代入上面的程序，就可以得到所求的打印前 $N$ 个素数的程序。但是，这个程序的效率是不高的。根据数论知识，可以从以下两点进行简化。

(1) 若 $x$ MOD $k$=0 且 $k>\sqrt{x}$ 可设 $x = k\times j$,显然 $j\leqslant\sqrt{x}$，即若 $x$ 被大于 $\sqrt{x}$ 的 $k$ 整除，必有一个不超过 $\sqrt{x}$ 的 $j$ 也能整除 $x$。因而，上界 lim 不必取为 $x$-1，只需取为 $\sqrt{x}$ 即可。

(2) 由于除了第一个素数 2 以外，其余的素数均为奇数。因而如果将 2 单独处理，那么，第二步中，语句 x:=x+1 可以改为 x:=x+2。

根据以上两点，语句 x:="下一个素数"，可以表示为

```
repeat
  x:=x+2; lim:=sqrt(x);
  k:=2; prim:=true;
  while prim AND(k≤lim)do
    begin
      prim:=(x MOD k)≠0;
      k:=k+1;
    end
until prim;
```

将这一程序段代入第一步给出的程序，就得到所求的程序。

第五步：上面得到的程序当 $N$ 较大时，计算量仍然比较大，可以进一步对程序进行加工。事实上，若 $x$ 能被 $k$ 整除，则必定为 $k$ 的素因子整除。因而，要确定 $x$ 是不是素数，只要检查 $x$ 能不能被不超过 $\sqrt{x}$ 的素数整除即可。为了方便，可设置数组 $p$，$p[k]$中存放第 $k$ 个素数，即 $p[1]=2$，$p[2]=3$，…。经过这样处理以后，上面的程序段可以改写为

```
repeat
  x:=x+2; k:=2; prim:=true;
  while prim AND(k≤lim) do
    begin
      prim:=(x MOD p[k])≠0;
      k:=k+1;
    end
until prim;
p[i]:=x;
```

由于这里的 $k$ 和第四步中的 $k$ 含义不同。因而，lim 应如何选取需要进一步考虑。和上面同样的道理，可取 lim 使

$$p[\text{lim}]>\sqrt{x} \text{ 而 } p[\text{lim-1}]\leqslant\sqrt{x}$$

或者

$$(p[\text{lim}])^2>x \text{ 而}(p[\text{lim-1}])^2\leqslant x$$

这样，对所有满足 $k<$lim 的 $k$，均有

$$p[k]\ \leqslant\sqrt{x}$$

由于随着 x 的改变，lim 在改变。因而可以利用条件语句在循环中产生 lim。综合这些想法，并补充上必要的类型说明和变量说明，就可以得到求解这个问题的一个完整的程序。

```
program ex1(output);
type index=1 •• N;
var x: integer;
   i,k, lim: index;
   prim:boolean;
   p: array [index] of  integer;
begin
    p[1]:=2; write[2];
    x:=1;     lim:=1;
    for i:=2 to N do
      begin
        repeat x:=x+2;
        if sqr(p[lim])<x then lim:=lim+1;
        k:=2;prim:=true;
        while prim AND(k<lim) do
          begin
            prim:=(x MOD p[k])≠0;
            k:=k+1;
          end
        until prim;
        p[i]:=x;   write(x);
      end
end.
```

通过上面这个简单的例子，可以对逐步求精的程序设计方法有一个粗略的了解。为了加深理解，下面再举两个比较复杂的例子。为了便于理解，仍然使用大家熟悉的 PASCAL 语言来编写程序。

**例 1.4**　对给定的自然数 $n$，确定满足下述关系的最小的数 $s$，$s$ 可表示为两对不同的自然数的 $n$ 次方之和，即找出最小的数 $s$，使得

$s=a^n+b^n=c^n+d^n$

这里，$a$、$b$、$c$、$d$ 是自然数，且 $a\neq c$，$b\neq d$。

例如，对于 $n$=2 的情形，可以计算出表 1.1。

表 1.1

| i \ j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 2 | | | | | | | |
| 2 | 5 | 8 | | | | | | |
| 3 | 10 | 13 | 18 | | | | | |
| 4 | 17 | 20 | 25 | 32 | | | | |
| 5 | 26 | 29 | 34 | 41 | 50 | | | |

(续)

| i \ j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 6 | 37 | 40 | 45 | 52 | 61 | 72 | | |
| 7 | 50 | 53 | 58 | 65 | 74 | 85 | 98 | |
| 8 | 65 | 68 | 73 | 80 | 89 | 100 | 113 | 128 |

由表 1.1 可知

$$50=1^2+7^2=5^2+5^2$$
$$65=1^2+8^2=4^2+7^2$$
$$85=2^2+9^2=6^2+7^2$$
$$\cdots$$

从而，所求的最小数 $s=50$。为了讨论方便起见，令 $s_{ij}=i^n+j^n$。显然，$s_{ij}$ 满足下面 3 个性质：

**性质 1**：$s_{ij}>s_{ik}$，对所有的 $i$ 及所有的 $j>k$。

**性质 2**：$s_{ij}>s_{kj}$，所有的 $j$ 及所有的 $i>k$。

**性质 3**：$s_{ij}=s_{ji}$，对所有的 $i,j$。

根据性质 3，只需考虑 $j<i$ 的情形（表 1.1 中仅给出对角线以下部分就是这个道理）。

如何用程序实现这个挑选过程呢？看来，似乎最直接的方法是：在上表中按先行后列的顺序逐步地生成每一个“候选者” $s_{ij}$，同时和已生成的候选者进行比较，直至遇到两个候选者相同为止，即可求得 $s$。但是，求得的 $s$ 是不是所要求的最小数据呢？这一点并没有证明。退一步讲，即使这样做是许可的，为了进行比较，必须保留所有已经生成的候选者，这无疑要占用较大的存储空间。之所以会产生这些问题，关键在于这时候选者不是按大小顺序排列的。如果我们不是按先行后列，而是按由小到大的顺序逐步地生成候选者 $s_{ij}$,那么可采用上面的方法求出 $s$。

下面，我们就 $n=3$ 的情形，采用逐步求精的方法编制解决这一问题的程序。

第一步：根据上面的一些基本想法，可得到程序

```
x:=2
repeat
  min:=x;
  x:="下一个较大的候选者"          (*)
until x=min;
write(min);
```

第二步：对语句（*）进一步求精。根据性质 1，每一行中后面的数总比前面的大，所以在挑选下一个较大的候选者的过程中，每行只需保留最后生成的一个候选者即可。为了表示语句（*），设置两个整型数组 $s$ 和 $j$，其中 $s[k]$表示第 $k$ 行中已经生成的最后一个后选取者，$j[k]$表示 $k$ 行中已经生成的最后一个后选者的下标 $j$。这样 $s[k]$，$j[k]$之间的关系为

$$s[k]=k^3+j[k]^3$$

如果再定义一个函数 $p(k)=k^3$，且用 $s[i]$代替 $x$，可得到程序

```
i:=1;
```

```
for k:=1 to ? do
  begin
    j[k]:=1; s[k]:=p(k)+1;
  end;
repeat
  min:=s[i];
1:“增大 j[i]，且用第 i 行中下一个候选者替换 s[i]”
2:“确定 i 的新值作为具有最小候选者那行的下标”
until s[i]:=min;
write(min);
```

第三步：在上面这段程序中，给 $j[k]$、$s[k]$赋初值的个数是不确定的，同时在执行语句 2 时，进行挑选的行的个数也是不确定的。考虑到 $s_{ij}$ 满足的性质 2，可选变量 ih 作为界限，即只对 $k \leqslant$ih 的行进行挑选和赋初值。ih 选取的方法是：ih 是使 $j[i]=1$ 成立的最小的 $i(i \neq 1)$。这样的 ih 可以在挑选候选者的过程中产生。上面的程序中赋初值部分可以简化，从而可得下面的程序

```
i:=1; ih:=2;
j[1]:=1; s[1]:=2; j[2]:=1;s[2]:=p(2)+1;
repeat
  min:=s[i]
  1:if j[i]=1 then “增大 ih，且给 s[ih]置初值”;
  2:“增大 j[i],且用 i 行中下一个候选者替换 s[i]”;
  3:“确定 i,使得 s[i]=min(s[1],…，s[ih])”
until s[i]=min;
write(min);
```

第四步：对带有标号 1～3 的语句进一步求精。首先，根据 $s_{ij}$ 满足的性质 3，我们知道应有 $j \leqslant i$，这样，在挑选过程若遇到 $j[i]=i$ 时，在 $i$ 行中不能再生成候选者，而且,这时候由于第 $i$ 行的候选者已全部生成，所以在执行语句 3 时就不需要再考虑这一行了。这也就是说在执行语句 3 时，$i$ 应该有一个下界 $i1$。每消掉一行时，$i1$ 就增加 1。这样，可得下面的程序

```
i:=1; ih:=2; i1:=1;
j[1]:=1; s[1]:=2; j[2]:=1; s[2]:=p(2)+1;
repeat
  min:=s[i];
  if j[i]=i then i1:=i1+1 else
      begin
      1:if j[i]=1 then“增大 ih，且给 s[ih]置初值”
      2:“增大 j[i],且用 i 行中下一个候选者替换 s[i]”
      end;
      3:“确定 i,使 s[i]=min(s[i],…，s[ih])”
until s[i]=min;
```